BEval: A Plug-in to Extend Atelier B with Current Verification Technologies

Valério Medeiros Jr. [*]
IFRN
Natal, Brazil
Federal Institute of Education, Science and Technology of Rio Grande do Norte
Natal, Brazil
valerio.medeiros@ifrn.edu.br

David Déharbe [*]
UFRN
Natal, Brazil
Federal University of Rio Grande do Norte
Department of Informatics and Applied Mathematics
Natal, Brazil
david@dimap.ufrn.br

This paper presents BEval, an extension of Atelier B to improve automation in the verification activities in the B method or Event-B. It combines a tool for managing and verifying software projects (Atelier B) and a model checker/animator (ProB) so that the verification conditions generated in the former are evaluated with the latter. In our experiments, the two main verification strategies (manual and automatic) showed significant improvement as ProB's evaluator proves complementary to Atelier B built-in provers. We conducted experiments with the B model of a micro-controller instruction set; several verification conditions, that we were not able to discharge automatically or manually with Atelier B's provers, were automatically verified using BEval.

1 Introduction

Classical B and Event-B are formal methods initially developed by J.-R. Abrial [1, 2] that contain the notion of abstract machine and refinement. These methods are widely applied in safety critical systems and supported by Atelier B [5] and others tools. ProB [8] is a tool for animation, model checking as well as an expression evaluator. A Rodin plug-in [2, 9] to interact with ProB has been developed and is used as a disprover. The goal of the BEval project was to develop a similar plug-in for Atelier B. This goal was driven by our attempt to streamline the verification of proof obligations generated in the development of a formal model of a micro-controller instruction set [10]. Indeed their verification with the automatic prover available in Atelier-B was often inconclusive and required time-consuming use of the interactive prover.

Currently, several components of Atelier B are neither open source nor free, most notably the mathematical rule validator tool and the theorem prover for the B method and Event-B are closed. Moreover the main Atelier B theorem prover (krt) did not evolve significantly in the past decades. Indeed, to develop (and sell!) safety-critical systems, tools, such as Atelier B, need to pass a costly certification process. Of course, this prevents continuous evolution of these components. However, recent development in verification technologies, such as other satisfiability modulo-theories (SMT) solvers [3, 11] has resulted in significant progress. Therefore, we consider the time is ripe to evaluate, through an open source project, the potential contribution of incorporating such technologies in the tool set. BEval is our contribution towards this goal: an Atelier B plug-in that provides additional verification engines and can be used for different utilities like: a disprover searching counterexamples [4], a theorem prover verifying the proof obligations and a mathematical rule validator tool checking new reusable rules.

[*]This work has been partially supported by CNPq grants 560014/2010-4, 308008/2012-0, 573964/2008-4 (National Institute of Science and Technology for Software Engineer - INES)

Besides, there are new requirements to the verification process in the B method. We present our case study that produces proof obligations that cannot be verified automatically with Atelier B built-in provers, and are very difficult to show interactively, probably because these involve complex expressions with bit vectors and math operators.

This paper is organized as follows. Section 2 presents BEval and its context. Then section 3 provides an experimental evaluation of BEval based on a case study. We conclude with remarks in the last section.

2 The BEval Plug-in

BEval[1] is an open source tool to systematize the verification of B expressions in Atelier B by integrating ProB. BEval allows one to select B expressions and submits them for evaluation with ProB. When the expression is true, BEval creates a matching proof rule compatible with Atelier B built-in provers which may then discharge them automatically.

2.1 BEval's Architecture

A B development generates a set of proof obligations. First, these proof obligations are analyzed with Atelier B built-in automatic provers. Then the remaining proof obligations can be evaluated individually either by Atelier B interactive prover or by BEval. The architecture of the verification framework augmented with BEval is presented in Figure 1. BEval provides two graphical user interfaces: a rule evaluator that analyzes only one proof obligation and a component evaluator that analyzes a set of selected proof obligations.

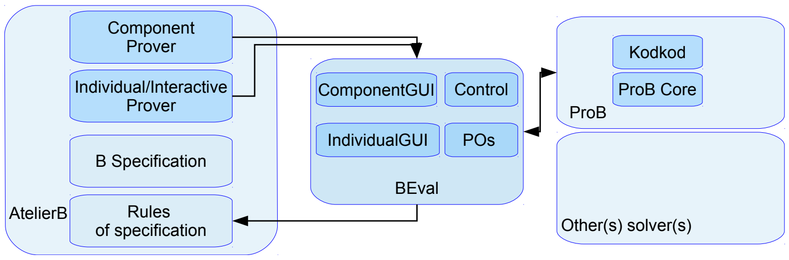


Figure 1: An overview of component architecture.

Internally, BEval is composed of different components. They are:

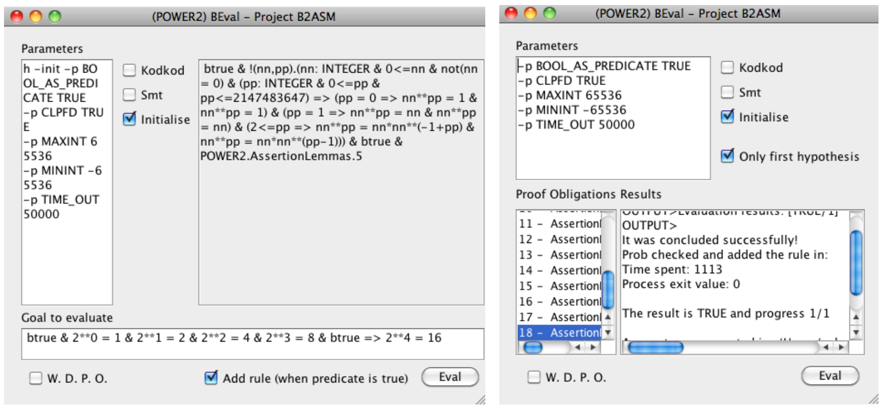
- `IndividualGUI` provides the graphical user interface for evaluation of an expression from within the interactive prover of Atelier B. This expression can be a full proof obligation, just a hypothesis of proof obligation, or a rule that embodies an important logic rule. Once verified, the tool offers to add the rule to the set of rules of the current specification in the interactive prover.
- `ComponentGUI` provides the graphical user interface for evaluation of a set of proof obligations. This graphical interface contains different text areas: ProB evaluation parameters; results of evaluations; a list of proof obligations with its current state (proved/unproved).
- `Control` is responsible for controlling the communication between the tools. The communication between Atelier B, BEval and ProB is simple and uses command-line arguments from Java.

[1]The BEval and its video demonstration are available at: https://github.com/ValerioMedeiros/BEval

Basically, Atelier B invokes a shell script passing as arguments information such as the path of module and the expression to evaluate. The shell script invokes BEval that redirects the output to a graphical user interface and calls a ProB client in the background.

- `POs` is responsible for managing the proof obligations stored in Atelier B format. This component is able to import one proof obligation or a set of proof obligations; and to export a set of true rules compatible with Atelier B.

2.2 Graphical User Interface for Evaluating Proof Obligations


(POWER2) BEval – Project B2ASM
Parameters
Goal to evaluate
Eval
(POWER2) BEval – Project B2ASM
Parameters
Proof Obligations Results
Eval

```
h -init -p BO
OL_AS_PREDI
CATE TRUE
-p CLPFD TRU
E
-p MAXINT 6
5536
-p MININT -6
5536
-p TIME_OUT
50000
```

☐ Kodkod
☐ Smt
☑ Initialise

```
 btrue & !(nn,pp).(nn: INTEGER & 0<=nn & not(nn
= 0) & (pp: INTEGER & 0<=pp &
pp<=2147483647) => (pp = 0 => nn**pp = 1 &
nn**pp = 1) & (pp = 1 => nn**pp = nn & nn**pp
= nn) & (2<=pp => nn**pp = nn*nn**(-1+pp) &
nn**pp = nn*nn**(pp-1))) & btrue &
POWER2.AssertionLemmas.5
```

Goal to evaluate: `btrue & 2**0 = 1 & 2**1 = 2 & 2**2 = 4 & 2**3 = 8 & btrue => 2**4 = 16`

☐ W. D. P. O. ☑ Add rule (when predicate is true)

```
-p BOOL_AS_PREDICATE TRUE
-p CLPFD TRUE
-p MAXINT 65536
-p MININT -65536
-p TIME_OUT 50000
```

☐ Kodkod
☐ Smt
☑ Initialise
☑ Only first hypothesis

```
11 - AssertionI
12 - AssertionI
13 - AssertionI
14 - AssertionI
15 - AssertionI
16 - AssertionI
17 - AssertionI
18 - AssertionI
```

```
OUTPUT>Evaluation results: [TRUE/1]
OUTPUT>
It was concluded successfully!
Prob checked and added the rule in:
Time spent: 1113
Process exit value: 0

The result is TRUE and progress 1/1
```

☐ W. D. P. O.

Figure 2: The graphical user interfaces of BEval: on the left, the graphical interface to submit one expression to evaluation; on the right, the graphical interface to submit a set of proof obligations.

Figure 2 shows the two graphical user interfaces contributed by BEval. The left graphical interface can be invoked from the interactive prover of Atelier B. It has the following elements:

- *Parameters* - located on the top-left of the window, it is an editable text where the user has access to the options used to call ProB;
- *Main options* - located on the top-middle of the window, three basic options are checkable; *Kodkod* indicates that ProB may use the tool of the same name; *Smt* indicates that ProB uses a more aggressive constraint solver; *Initialise* indicates that definitions from the B component shall be loaded;
- *Hypothesis* - located on the top-right of the window, it presents the hypothesis that the user may want to add to the goal, addition of such hypothesis shall be performed with copy-and-paste operations;
- *Goal to evaluate* - it is an editable text that contains the expression that will be sent to ProB;
- *Add rule* - if this option is checked, whenever the goal evaluates to "true", a rule is generated and added into the corresponding `pmm` file [2];
- *W.D.P.O.* - if that option is checked, then, whenever the goal evaluates to "true", the generated rule will be added to the corresponding `wd_pmm` [3], otherwise it will be added to the common `pmm` file.

[2]Atelier-B associates to each project a `pmm` file, where additional proof rules may be stored for use by the automatic provers to discharge proof obligations.

[3]The `wd_pmm` file has a role similar to the `pmm` file but is used to discharge well-definedness proof obligations.

- *Eval* - this button provokes the call to ProB on the current goal with the given list of parameters.

The right graphical interface is used within the components window. It is similar, but has additional elements. First, *Proof obligations* contains a list of proof obligations and only those selected are evaluated. By default, the selected items are unproved proof obligations. Second, *Result* is a text area that contains the output results of ProB's evaluations. The remaining buttons are related to ProB parameters and are explained in the next section.

2.3 Evaluation Parameters

The options and parameters to the verification process are crucial and the following are used by default in BEval:

- **-p MAXINT 65536 -p MININT -65536** sets the range for integers.
- **-p init** loads definitions from B module. This parameter is useful when the proof obligation was not fully expanded in only logic and math definitions. This parameter must be used when the proof obligation has a dependency of definitions. For example, the proof obligation "$[0,0,0,0,0,0,0,0] \in BYTE$" depends on the definition $BYTE = (1..8 \rightarrow \{0,1\})$, so that this proof obligation is expanded to "$[0,0,0,0,0,0,0,0] \in (1..8 \rightarrow \{0,1\})$" and becomes independent.
- **-p KODKOD TRUE** indicates that ProB may use a constraint solver for relational logic, called Kodkod. This parameter allows a mixture of SAT-solving and ProB's own constraint-solving capabilities according to [12].
- **-p TIME_OUT** sets the run time budget for the constraint solver.
- **-p SMT TRUE** forces ProB to do more aggressive constraint solving.
- **-p CLPFD TRUE** enables constraint logic programming over finite domains. It restricts range to $(-2^{28}..2^{28-1})$ on 32 bit computers.

Additional parameters are available, the full list being available in ProB's web site [7].

2.4 Adding Rules

A rule is a formula added as an axiom in the prover's theories by being stored in a `pmm` file associated to a B component. The created rules can be reused to solve several proof obligations. These rules can be added individually by `IndividualGUI` or several rules can be added by `ComponentGUI`. The following rule is very simple and it was generated by BEval containing the information: name, date, spent time and the rule composed by hypothesis and goal.

```
THEORY RulesProBAssertionLemmas_1 IS
  /* Expression from (AssertionLemmas_1), it was added  in Thu Jun 27 18:02:32 BRT 2013
  evaluated with ProB in 5913 milliseconds. Module Path:/B_Resources/BYTE_DEFINITION.mch */
  "'Check assertion (card(BYTE) = 256) deduction - ref 3.2, 4.2, 5.3'"
  BYTE = (1..8 --> {0,1}) =>   (card(BYTE) = 256)
END
```

Each created true rule has a relation with one proof obligation. When the interactive prover of Atelier B is evaluating a proof obligation and BEval-`IndividualGUI` created a rule then the interactive prover can apply the created rule in the evaluation of the current proof obligation.

BEval-ComponentGUI creates a set of rules and a set of "*User Pass*", that is a sequence of proof commands. A *User Pass* can be used in automatic prover and can indicate a rule to apply in the selected proof obligation. Each created *User Pass* selects the proof obligations by name and defines the rule to invoke. The following *User Pass* example selects the proof obligation named of *initialisation* and invokes the prover using the rule named of "Rule1".

```
THEORY User_Pass IS
        Operation(Initialisation) & mp(Tac(RulesProBAssertionLemmas_1))
END
```

3 Experiments

We used Atelier-B to develop a reusable set of basic definitions to model hardware concepts, data types concepts and a micro-controller instruction set [10]. These definitions are grouped into separated development projects and are available as libraries.

The following table presents only the results of the most basic components using the default parameters of BEval. The components *Power* and *Power2* contain the standard definition of exponentiation and it is essential to establish the relationship between bit vectors and integer arithmetics. The components *BIT*, *BYTE* and *BV16* define bit, bit vectors with size 8 and 16, basic functions to manipulate bit vectors and important lemmas.

The proof obligations are classified in two groups: common and W.D. (well-definedness proof obligations). The columns represent respectively: **T. POs**, total number of proof obligations; **F1**, number of verified proof obligations with force 1 of Atelier B's prover; **F1;F2;F3**, number of verified proof obligations with force 3[4] after applying force 1 and 2; **F1;F2;F3;BEVAL**, number of verified proof obligations with BEval after applying forces 1, 2 and 3; **Gain**, percentage of proof obligations verified automatically and exclusively by BEval. The symbol "-" represents no changes in the number of verified proof obligations compared to the last applied strategy.

	Common POs					**W. D. P. Os.**				
Name	**T. POs**	**F1**	**F1;F2;F3**	**F1;F2;F3;BEval**	**Gain**	**T. POs**	**F1**	**F1;F2;F3**	**F1;F2;F3;BEval**	**Gain**
Power	3	2	-	-	0%	4	4	-	-	0%
Power2	18	2	-	18	88%	0	-	-	-	0%
BIT	49	23	-	49	53%	69	30	-	-	0%
BYTE	18	12	-	18	33%	136	129	-	132	2%
BV16	6	2	-	6	66%	69	67	-	69	2%

Almost all components have significant gains using BEval. This is significative since it relieves the developer from the burden of manually verifying a significant percentage of proof obligations and helps him focus on the more interesting proof obligations and ultimately benefits his productivity.

However, there are still some issues and limitations. Differences in the B syntax supported by Atelier B and ProB need to be fixed to support the evaluation of all components of micro-controller [10].

4 Conclusion and Perspectives

Finally, BEval is a tool able to import proof obligations from Atelier B, and convert and submit them for evaluation to ProB, interprets the results of that evaluation and create proof rules in Atelier-B ac-

[4]Higher forces use mechanisms consuming more time, CPU and memory resources.

cordingly. BEval's integration allows to exploit different strategies from the theorem prover of Atelier B and constraint logic solver of ProB. The results obtained with the verification of hardware library demonstrates a better ability of the constraint logic solver of ProB than the theorem prover of Atelier B for manipulating a class of expressions. The results presented in this paper show again that providing a port-folio of complementary provers is an effective approach to improve IDEs for formal development.

Another related tool is the Rodin SMT Plug-in [6], this plug-in supports proof obligations generated from event-B specifications and converts them to SMT format. In the future, BEval can also be integrated to Rodin SMT Plug-in and exploit its abilities. Alternatively, the current SMT translator of ProB [12] can be improved and integrated with news SMT solvers.

There are several possible new features and improvements for BEval. The small differences in B parsers of Atelier B and ProB can be solved by creating a pre-parser. Besides, ProB also has some limitations related to B constructs supported by Atelier B, but these limitations are being solved; also other tools may be investigated.

Acknowledgment. The evaluation of the model would not have been possible without the help of Michael Leuschel who kindly provided feedback and developed improvements to ProB to meet our needs.

References

[1] Jean-Raymond Abrial (2005): *The B-book - assigning programs to meanings.* Cambridge University Press.

[2] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta & Laurent Voisin (2010): *Rodin: an open toolset for modelling and reasoning in Event-B. STTT* 12(6), pp. 447–466. Available at `http://dx.doi.org/10.1007/s10009-010-0145-y`.

[3] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds & Cesare Tinelli (2011): *CVC4.* In: *CAV*, pp. 171–177. Available at `http://dx.doi.org/10.1007/978-3-642-22110-1_14`.

[4] Jens Bendisposto, Michael Leuschel, O. Ligot & Mireille Samia (2008): *La validation de modèles Event-B avec le plug-in ProB pour RODIN. Technique et Science Informatiques* 27(8), pp. 1065–1084. Available at `http://dx.doi.org/10.3166/tsi.27.1065-1084`.

[5] ClearSy System Engineering, Aix-en-Provence: *Atelier B - User Manual.* Available at `http://www.atelierb.eu/manuels/manuel-utilisateur-atelier-b-4.0-en.pdf`.

[6] David Déharbe, Pascal Fontaine, Yoann Guyot & Laurent Voisin (2012): *SMT Solvers for Rodin.* In: *ABZ*, pp. 194–207. Available at `http://dx.doi.org/10.1007/978-3-642-30885-7_14`.

[7] Michael Leuschel (2011): *User Manual.* Available at `http://www.stups.uni-duesseldorf.de/ProB/index.php5/User_Manual`.

[8] Michael Leuschel & Michael J. Butler (2003): *ProB: A Model Checker for B.* In: *FME*, pp. 855–874. Available at `http://dx.doi.org/10.1007/978-3-540-45236-2_46`.

[9] Olivier Ligot, Jens Bendisposto & Michael Leuschel (2007): *Debugging Event-B Models using the ProB Disprover Plug-in. Proceedings AFADL'07.*

[10] Valério G. Medeiros Jr. & David Déharbe (2012): *Experience in Modeling a Microcontroller Instruction Set Using B.* In: *Brazilian Symposium on Formal Methods*, SBMF, Natal - RN.

[11] Leonardo Mendonça de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver.* In: *TACAS*, pp. 337–340. Available at `http://dx.doi.org/10.1007/978-3-540-78800-3_24`.

[12] Daniel Plagge & Michael Leuschel (2012): *Validating B, Z and TLA + Using ProB and Kodkod.* In: *FM*, pp. 372–386. Available at `http://dx.doi.org/10.1007/978-3-642-32759-9_31`.